JS Advance

Assignment by reference

Please take out your paper again and do a T-diagram exercise, on your paper, to predict the output for the following code. It's important that you go through this slowly one by one.

Exercise 1

```
let a = 5;
let b = a;
a = 100;
console.log('a is now', a);
console.log('b is now', b);
```

Exercise 2

```
let a = [5];
let b = a;
a.push(100);
console.log('a is now', a);
console.log('b is now', b);
```

Are you surprised both a and b are now [5,100]? Isn't that weird? The only thing that was changed was the value in a. How did b also get changed?

This is an important concept to understand and it's called an assignment by reference.

More about assignment by reference

If you're creating a string or a number and storing these in a variable, these get created in what's called a call stack. Whenever you store an array, an object, a function, or a class in a variable, these values get created in what's called a heap stack. Why computer program does assignment this way requires a bit more explanation on how computer was designed, which is discussed in the video.

For example,

```
let a = 5;  //creates this value in the call stack
let b = a;  //duplicates the value stored in a and assigns that to b
b = b + 5;  //b is updated to be 10.
console.log(a, b); //logs 5 and 10
let c = "Michael";  //creates this value in the call stack
let d = c;  //duplicates the value stored in c and stores that to d
d = d + " Choi";  //d is updated as "Michael Choi"
console.log(c,d); //logs "Michael" and "Michael Choi"
let x = [1,3,5]; //creates this value in the heap stack
let y = x;  //y also points to that value in the heap stack
y.push(7); //at where y is pointed to, it pushes a new value called 7
console.log(x, y); //logs [1,3,5,7] twice!
```

References:

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/Array
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/Object
- https://www.sitepoint.com/how-javascript-references-work/
- https://stackoverflow.com/questions/79923/what-and-where-are-the-stack-and-heap
- https://www.geeksforgeeks.org/stack-vs-heap-memory-allocation/

Immediate Function

Imagine that you have the following code in your javascript library, say hero.js.

In that file, you had the following code.

```
var x = 45;
//
function sum(a, b) {
  console.log('hero.js | adding two numbers to x');
  return x+a+b;
}
```

Imagine that your javascript library was used by a lot of developers and these developers not only used just your javascript library but also used multiple other javascript libraries.

For example, imagine that one front-end developer, had the following html:

```
<html>
<head>
   <title>Awesome App</title>
   <script src="hero.js"></script>
   <script src="jquery.js"></script>
   <script src="bootstrap.js"></script>
   <script src="react.js"></script>
   <script src='underscore.js"></script>
   ...
   <script src='meteor.js"></script>
...
```

Say that either in another javascript library in their own javascript code, they also had a variable called x and a function called sum. For example, say that in one of the javascript file that was loaded, it had something like this:

```
var x = "School";
function sum(...) {
  ...
}
```

If this is the case, how would the T-diagram for the application look like? Would it still remember the x values from both of these libraries? What about the function sum?

The answer is that the application won't remember both as they are all named the same! After all, in the T-diagram, if there was a spot for storing what is stored in x, it can only store one value and not two values, right?

This conflict of variable names or function names is what's called a namespace conflict. It can lead to some funny behavior as you may be calling a function 'sum' thinking it would do one thing (what was outlined in hero.js) but where you find out it was somehow over-written by another javascript library that you loaded that also had the same function name, etc.

This can cause a lot of nightmare and countless hours of debugging. This is not fun.

So how could you structure your code now so that these variable names or functions you've written in these libraries do NOT conflict with one another?

One way to do this is by using Classes. If you use either a ES5 Class/Function or a ES6 Class, you don't have to worry about these name conflicts as all the variables and methods are encapsulated inside the function or a class. There is however another way to fix this problem even if you weren't using a class.

For example, consider the following:

```
let a = 35;
let x = "Doritos";
function abc(...){
  //things that do wonderful stuff with a, x, and some other stuff
  return ...
}
function edf(...){
  //things that do wonderful stuff with variables a, x, and some other stuff
  return ...
}
...
```

Above is bad as variable x, y and function abc, and edf are very common and your variables/methods could be over-written by someone else, or even worse, break other people's codes.

So what if you did this instead?

```
var unique_name = (function(){
   let a = 35;
   let x = "Doritos";
   function abc(...){
      //things that do wonderful stuff with a, x, and some other stuff
      return ...
   }
   function edf(...){
      //things that do wonderful stuff with variables a, x, and some other stuff
      return ...
   }
   return { abc: abc, edf: edf };
})();
```

So what's going on here?

What happened above was that you put all the code inside a function, then you immediately executed that function! All the variables and methods now live inside **unique_name** and you don't have to worry about the troublesome name conflicts you had to deal with.

Now, what if in this library, or basically inside the unique_name library, you needed to access some global variables such as the window object, or the jQuery object, or the document itself?

You could simply pass these objects as arguments to the immediate function by doing this:

```
var unique_name = (function(window, jQuery, document){
   let a = 35;
   let x = "Doritos";
   function abc(...){
      //things that do wonderful stuff with a, x, and some other stuff
      return ...
   }
   function edf(...){
      //things that do wonderful stuff with variables a, x, and some other stuff
      return ...
   }
   return { abc: abc, edf: edf };
})(window, jQuery, document);
```

Now inside that function, you can use these variables freely! Isn't that cool? So now, you've protected your code from other potential variable/method name conflicts and as long as other libraries do not use a variable name that's identical to your first line, you'll be fine!

In summary, an immediate function is a function that's invoked immediately. For example,

```
(function(){
    //some code...
   function abc() {
     //additional code.
   }
})();
```

As we discussed, you could also pass other variables into this immediate function. You'll note that a lot of javascript libraries such as jQuery use immediate functions to encapsulate their methods/variables. All good Javascript developers know about an immediate function, so you should also. :)

To read more about immediate function, you could try the following resources. Do however limit your read to no more than 10 minutes (for reading all of these articles). Sometimes, these concepts aren't really explained the way that's easy to understand so if you understood what an immediate function is, by reading what I wrote above, be satisfied with your knowledge and move on, knowing that any of these concepts, if we go more in depth, could start getting more complex also.

Additional resources you could try reading (but cap your reading to be no more than 10 minutes):

- https://developer.mozilla.org/en-US/docs/Glossary/IIFE
- https://stackoverflow.com/questions/939386/immediate-function-invocation-syntax
- https://blog.kevinchisholm.com/javascript/javascript-immediate-functions-basics/
- https://medium.com/@vvkchandra/essential-javascript-mastering-immediately-invoked-function-expressions-67791338ddc6

Closures

A closure is basically when there is a function inside a function. Because of how Javascript handles the scoping, when a function exists inside a function, the inner function tends to, in a way, 'remember' the T-diagram of the parent function.

This term, 'closures' is used quite a lot and when you read other documentations, you'll probably get exposed to this term. There are much more complex explanations about what a closure is and how this variable scoping works inside the inner function, but for now just remember that if you focus on the fundamentals and remember always to do a T-diagram, you won't struggle that much understanding closures.

For example, let's take a look at the following:

```
function abc(){
   var name = "Michael";
   function sayHello(){
      console.log(name);
   }
   sayHello();
}
abc();
```

Run this using paper and try to predict what the output would be, making sure you use a T-diagram.

Now what about the following?

```
function abc(){
   var name = "Michael";
   function sayHello(){
      console.log(name);
   }
   return sayHello;
}
let result = abc();
console.log(result);  //what would get logged here???
//
console.log(result()); //what would get logged now?
```

Run this using paper again using a T-diagram and making sure that () is needed to execute a function stored in a variable. Are you surprised that this still works?

Note that when sayHello was returned, it returned a whole function. In that function, it contained instruction for 'console.log(name)' although it did not execute that instruction just yet. Later when that function was executed, by the last line, however, how did the function remember what was stored in the variable called name???

This nature where the function inside the function still remembers variables of the parent function, even when the program left/existed the parent function is called closures.

With the ES5/ES6 OOP syntaxes, you probably won't have to deal that much with closures. However, as closures make great interview questions, especially to identify casual JS developers vs serious JS developers, we wanted you to be familiar with this term and not get your confidence blown just because you haven't learned about this terminology.

Asynchronous Callbacks

Imagine that you have the following code. Using a physical paper, run each line one by one and try to predict the output of the code. Do not skip doing this but do this manually on paper first. This is an important concept and you need to do this slowly. To make it easier for you to read the code, I've included a line number.

Exercise 1

```
1) let a = (function task1() {
2)     $.get('/home', function(res) {
3)         $('#main').html(res);
4)         console.log('hello');
5)         return 100;
6)     }
7)     console.log('world');
8) })();
9) console.log(a);
```

I've used immediate function above to immediate execute the function that was designed. What would get logged when the computer reaches the end?

Now run the code and see what the output is. Are you surprised?

Exercise 2

```
1) let a = (function task1() {
2)     $.get('/home', function(res) {
3)         $('#main').html(res);
4)         console.log('hello');
```

```
5)         return 100;
6)     }
7)     console.log('world');
8)     return 5;
9) })();
10)console.log(a);
```

Exercise 3

```
1) let a = (function task1() {
2)     function abc() {
3)         console.log('hello');
4)         return 100;
5)     }
6)     abc();
7)    console.log('world');
8)     return 5;
9) })();
10)console.log(a);
```

Exercise 4

```
let a = (function() {
    return function(a, b) {
        console.log('hello');
        return a*b;
    }
    console.log('world');
})();
console.log(a);
```

Exercise 5

```
let a = (function() {
    return function(a, b) {
        console.log('hello');
        return a*b;
    }
    console.log('world');
})();
console.log(a());
```

Exercise 6

```
let a = (function() {
    return function(a, b) {
        console.log('hello');
        return a*b;
    }
    console.log('world');
})();
console.log(a(3,5));
```

Event handling of Javascript

Imagine that you wanted to write your program such that the following tasks are performed in sequence.

- Send an Ajax request to a url A
- If the Ajax request to a url A was successful, then update some of the HTML DOM and then send another Ajax request to a url B.

- If the Ajax request to a url B was successful, then update some of the HTML DOM and then send another Ajax request to a url C.
- If the Ajax request to a url B was successful, then update some of the HTML DOM.

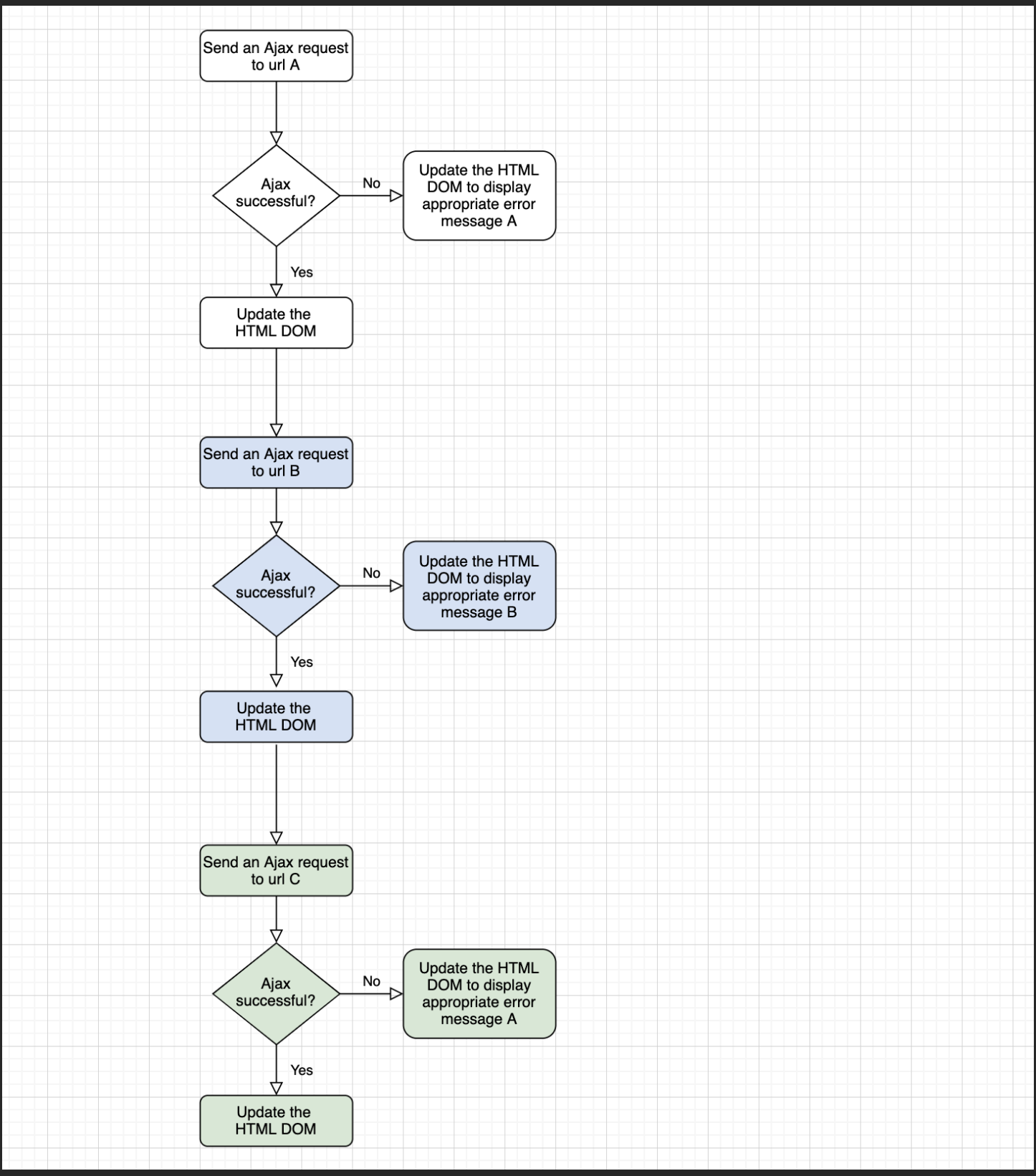
Seems straight forward right? Now, how would you write this in Javascript? Would this code work the way you wanted?

```
$.post('___url A___', function(res){
    //codes for updating the DOM after an Ajax request to url A was done successfully
    //codes for updating the DOM after an Ajax request to url A was done successfully
    //codes for updating the DOM after an Ajax request to url A was done successfully
    //codes for updating the DOM after an Ajax request to url A was done successfully
    //codes for updating the DOM after an Ajax request to url A was done successfully
    //codes for updating the DOM after an Ajax request to url A was done successfully
    //codes for updating the DOM after an Ajax request to url A was done successfully
    //codes for updating the DOM after an Ajax request to url A was done successfully
    //codes for updating the DOM after an Ajax request to url A was done successfully
    //codes for updating the DOM after an Ajax request to url A was done successfully
    //codes for updating the DOM after an Ajax request to url A was done successfully
    //codes for updating the DOM after an Ajax request to url A was done successfully
    //codes for updating the DOM after an Ajax request to url A was done successfully
    //codes for updating the DOM after an Ajax request to url A was done successfully
    //codes for updating the DOM after an Ajax request to url A was done successfully
    //codes for updating the DOM after an Ajax request to url A was done successfully
    //codes for updating the DOM after an Ajax request to url A was done successfully
    //codes for updating the DOM after an Ajax request to url A was done successfully
    //codes for updating the DOM after an Ajax request to url A was done successfully
    //codes for updating the DOM after an Ajax request to url A was done successfully
    //codes for updating the DOM after an Ajax request to url A was done successfully
    //codes for updating the DOM after an Ajax request to url A was done successfully
    //codes for updating the DOM after an Ajax request to url A was done successfully
});
$.post('___url B___', function(res){
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
    //codes for updating the DOM after an Ajax request to url B was done successfully
});
$.post('___url C___', function(res){
```

```
    //codes for updating the DOM after an Ajax request to url C was done successfully
    //codes for updating the DOM after an Ajax request to url C was done successfully
    //codes for updating the DOM after an Ajax request to url C was done successfully
    //codes for updating the DOM after an Ajax request to url C was done successfully
    //codes for updating the DOM after an Ajax request to url C was done successfully
    //codes for updating the DOM after an Ajax request to url C was done successfully
    //codes for updating the DOM after an Ajax request to url C was done successfully
    //codes for updating the DOM after an Ajax request to url C was done successfully
    //codes for updating the DOM after an Ajax request to url C was done successfully
    //codes for updating the DOM after an Ajax request to url C was done successfully
    //codes for updating the DOM after an Ajax request to url C was done successfully
    //codes for updating the DOM after an Ajax request to url C was done successfully
    //codes for updating the DOM after an Ajax request to url C was done successfully
    //codes for updating the DOM after an Ajax request to url C was done successfully
    //codes for updating the DOM after an Ajax request to url C was done successfully
    //codes for updating the DOM after an Ajax request to url C was done successfully
    //codes for updating the DOM after an Ajax request to url C was done successfully
    //codes for updating the DOM after an Ajax request to url C was done successfully
    //codes for updating the DOM after an Ajax request to url C was done successfully
    //codes for updating the DOM after an Ajax request to url C was done successfully
    //codes for updating the DOM after an Ajax request to url C was done successfully
    //codes for updating the DOM after an Ajax request to url C was done successfully
    //codes for updating the DOM after an Ajax request to url C was done successfully
    //codes for updating the DOM after an Ajax request to url C was done successfully
    //codes for updating the DOM after an Ajax request to url C was done successfully
    //codes for updating the DOM after an Ajax request to url C was done successfully
    //codes for updating the DOM after an Ajax request to url C was done successfully
});
```

Would this work the way you wanted? The answer is a no. Can you explain clearly why above wouldn't work the way you wanted? Above are common mistakes made by rookie Javascript developers, which a lot of times are caused because they have a hard time understanding exactly how callback functions work.

Now, let's make the program a bit more complex. Below is a slight modification to the logic presented above. This is still considered a very simple operation, something that lots of web applications will do things that are easily 3-10 times more complex than what was described. However, to make this learning quick and efficient, let's just look at this scenario and analyze.



How would you write a program to do this?

Arrow function

Arrow function was introduced in ES6 to make Javascript even more concise. However, I personally don't like my developers using arrow functions as I don't think it makes the code readable.

Please spend no more than 10 minutes reading this reference: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions.

In essence, whenever you're using a function (which is a lot of Javascript), you could remove having to type the word function, return, or use the curly brackets. Again, I personally think these words, although it may add 1-2 more lines, is worth it, but as you may see other developers using arrow functions, I want you to be familiar with what it is, and be able to translate what's done in arrow function to a normal function.

For example, note that

```
//traditional function
function abc(a) {
    return a+100;
}
//one way of using an arrow function
(a) => {
    return a+100;
}
//if the function is simply returning a value, the curly brackets and the return statement is optional
(a) => a+100;
//the parenthesis is even optional!
a => a+100;
```

As a challenge, and for you to get comfortable translating arrow functions into a traditional function, please convert the following arrow functions to traditional functions.

```
(a, b) => {
    let x = 10;
    return x+a*b;
}
(a=10, b=5) => a+b;
var obj = {
   name: 'coding';
   b: (word) => {
      console.log(this.name, word);
   }
}
obj.b('is fun');
```

Promises (still writing this)

Everything you can imagine for a web application, you could build things with what you've learned above. For handling a lot of events and what to do when certain event occurs, you can use 'callbacks'. For example, say that for the web

application, you're listening for certain event and if event A happens you want to do plan A and when event B happens you want to do plan B.

The way these are done are by using callbacks. For example, you say to the browser, "when event A happens, here is a function, with list of instructions on what to do next, that you can execute". The browser will listen for that event and when that event happens, the browser will execute that callback function you provided.

For years, this is what Javascript developers did. They used callbacks extensively to handle events and provided through callbacks series of instructions the browser should follow when event A, B, or C happens.

As Javascript became popular and as a lot more developers jumped in, a lot of these developers had a hard time understanding what callbacks were and how to use them effectively. Some used callbacks inefficiently or just didn't quite understand how to use it properly and produced codes that did not really make sense. They also struggled on writing codes this way, especially if they had programming background that was more centered around strong OOP languages such as PHP, C, Python, Java, etc.

The Javascript community adapted and they introduced this concept called Promises. This was in a way, just like how ES5 and ES6 were used to make Javascript look more like a OOP language, an effort to have developers write code in a way that looked more like other traditional languages, and also in a way, significantly improved the readability of the Javascript code, especially if the application was handling a lot of events.

For example, if you weren't that familiar with callbacks, what would you think would happen with the code below?

```
1) function abc(){
2)   sendAjaxRequest("www.cnn.com", function(res) {
3)      //callback instructions with what to do if the ajax request was successful
4)      console.log(1);
5)   }, function(res) {
6)      //callback instruction on what to do if the ajax request was not successful
7)      console.log(2);
8)   });
9)
10)   document.querySelector('h2').innerHTML = "Updated h2 tag!";
11)   //bunch of other instructions
12)   console.log(3);
13) }
```

Pull out your paper and do a T-diagram. If you looked at the code carefully, you'll see that the computer will log 3 first. Then it will log either 1 or 2. If you understood how a callback works, this will be natural and you'll know exactly what is happening and why it is happening. If not, pull out your paper and do a T-diagram and you'll see. Whenever a callback function is passed, it's not being executed immediately. Whoever receives the callback function can execute the callback function whenever they want and often these callback functions are executed much later. Therefore, the

Javascript interpreter will output 3 first as the other callback functions were not yet executed by sendAjaxRequest method.

Okay, so what if you wanted the instructions in line 10-12 to be executed AFTER the ajax request was sent? In other words, you want the program to WAIT and then run instructions line 10-12.

If you wanted to do this before Javascript introduced Promises (or Async/Await), there was not a way to do this. You just had to get better with learning how to use Callbacks and do this the Javascript way.

For example, the Javascript way would have been:

```
1) function abc(){
2)   sendAjaxRequest("www.cnn.com", function(res) {
3)      //callback instructions with what to do if the ajax request was successful
4)      console.log(1);
5)      document.querySelector('h2').innerHTML = "Updated h2 tag!";
6)      //bunch of other instructions
7)      console.log(3);
8)   }, function(res) {
9)      //callback instruction on what to do if the ajax request was not successful
10)      console.log(2);
11)   });
12) }
```

But if the instructions in line 6 were super long, and instructions for line 9 were super long, and not just a single line but say 50-200 lines, you can see how difficult reading through the code could be.

Promises were introduced to make the code more readable.

References

Please take up to 30 minutes to read through the following two articles. Whenever you see for example the Javascript having .then(...) statement, most likely it's using promises. Note that promise is not really required as everything could be done in callbacks but it does bring the benefit of making the code more readable.

However, with async and await, I personally believe that promises make code more confusing and it's a lot better to use async/await instead. As some Javascript developers as well as a lot of libraries you may be using, may still be leveraging promises, still get to learn what promises are by reading the articles below. Do spend no more than 30 minutes reading through these two articles though.

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise
- https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Promises

Async and Await

Go ahead and spend up to 15 minutes reading about async and await.

This was introduced as part of ECMAScript 2017 (and therefore is quite new). A lot of older Javascript libraries and browsers may not support async and await yet, although the latest node.js will support this feature. The browser, on the front end, may not support this feature yet, which is why it's safer to use this with Node.js but not yet on the front-end.

A good article for you to read about async and await is here:

- https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Async_await

Real World Example

For example, for Hacker Hero, when we were using callbacks, our model method looked as follows:

```
games.reorderGame = (post_data, result) => {
    connection.query(`UPDATE games_order SET game_ids_order = ? WHERE id = 1`,
    [post_data.order_list], (err, update_games_result) => {
        if(!err)
            result(err, {status : true, result : update_games_result})
        else
            result(err, {status: false, result : err});
    })
}
```

Note that we're passing a callback function as an argument to the query method.

When we re-factored the code to use async and await, it was changed to look as follows:

```
games.reorderGame = async (post_data, req=undefined) => {
    let databaseQueryModel    = new DatabaseQueryModel(req);
    let updateGameOrdersQuery = mysql.format(
        `UPDATE settings SET game_ids_order = ? WHERE id = 1`,
        [post_data.game_ids_order]
    );
    try {
        let result = await databaseQueryModel.executeQuery("Games Model | updateGameOrders", u
pdateGameOrdersQuery);
        return { status : true, result : result} ;
    } catch(err) {
        return { status: false, message: "Failed to reorder games", err: err} ;
    }
}
```

Note how using async and await makes the code a bit more readable.

The code-readability improves significantly as the logic becomes more complex and as you need to do things based on series of events. For example, take a look at the code below:

```
games.fetchAdminGames = (result) => {
    connection.query(`
    SELECT games.*, heroes.hero_name, companies.name as company_name, difficulty_levels.level
as level_difficulty_name , sublevels.challenge_ids_order,
    games.level_ids_order as subtopic_ids, count(challenges.id) as challenge_ids_order
    FROM games
    LEFT JOIN difficulty_levels ON difficulty_levels.id = games.difficulty_level_id
    LEFT JOIN heroes ON heroes.id = games.hero_id
    LEFT JOIN companies ON companies.id = games.company_id
    LEFT JOIN levels ON find_in_set(levels.id, games.level_ids_order)
    LEFT JOIN sublevels ON find_in_set(sublevels.id, levels.sublevel_ids_order)
    LEFT JOIN challenges ON challenges.sublevel_id = sublevels.id AND challenges.language_id =
1 AND challenges.challenge_type_id != 3
    LEFT JOIN challenge_groups ON challenge_groups.id = challenges.challenge_group_id
    JOIN games_order ON find_in_set(games.id, games_order.game_ids_order)
    WHERE (games.is_archived IS NULL OR games.is_archived = 0)
    AND (challenge_groups.is_archived IS NULL OR challenge_groups.is_archived = 0)
    GROUP BY games.id
    ORDER BY find_in_set(games.id, games_order.game_ids_order);
    `, (err, priority_order_games) => {
        if(err)
            result(err, {status : false, message : "Failed to fetch games."});
        else{
            let all_subtopics = [];
            let fetched_games = [];
            priority_order_games.map(game =>{
                if(game.subtopic_ids){
                    all_subtopics.push(game.subtopic_ids);
                }

                game.total_challenges = game.challenge_ids_order;
                fetched_games.push({...game, total_subtopics : 0 });
            });
            if(all_subtopics.length > 0){
                connection.query("SELECT * FROM levels WHERE levels.is_archived IS NULL  AND f
ind_in_set(levels.id, '"+ all_subtopics.join(',') +"')", (st_err, subtopic_response) => {
                    if(err)
                        result(err, {status : false, message : "Failed to fetch games."});
                    else{
                        if(subtopic_response){
                            fetched_games = priority_order_games.map(game =>{
                                let game_levels = (game.level_ids_order) ? game.level_ids_orde
r.split(',') : [];
                                game_levels = game_levels.map(game_level =>{
                                    return parseInt(game_level);
                                });
                                let game_subtopics = subtopic_response.filter(subtopic => {
                                    return (game_levels.indexOf(subtopic.id) > -1);
                                });
                                return { ...game, total_subtopics : game_subtopics.length };
                            });
                        }
                        result(err, {status : true, result : fetched_games});
                    }
                });
            }
            else
                result(err, {status : true, result : fetched_games});
        }
    })
}
```

This was re-factored and also using async/await, we got out of the callback hell behavior. This was the re-factored code afterwards.

```
// function used to fetch all the games or a specific game for the admin dashboard/chllanges p
ageQuery
// refactored by MC on June 2020
GetAdminGames = async (game_id = undefined, req=undefined) => {
    let databaseQueryModel       = new DatabaseQueryModel(req);
    let fetchAdminGamesquery = mysql.format(`
        SELECT
            games.id, games.game_type_id, games.title, games.description, games.is_beta, games
.is_visible, games.cache_players_count, games.cache_ratings_json,
            games.cache_challenges_count_json, games.thumbnail_url, games.game_url, games.intr
oduction, games.language_ids_order, heroes.hero_name, companies.name as company_name,
            games.level_ids_order as subtopic_ids,
            count(levels.id) as total_subtopics, FIND_IN_SET(games.id, (SELECT game_ids_order
FROM settings)) as game_order
        FROM games
        INNER JOIN heroes ON heroes.id = games.hero_id
        INNER JOIN companies ON companies.id = games.company_id
        LEFT JOIN levels ON levels.game_id = games.id AND levels.is_archived = 0
        WHERE games.is_archived = 0
        GROUP BY games.id
        ORDER BY game_order ASC;`
    );
    if(game_id != undefined){
        fetchAdminGamesquery = mysql.format(
            `SELECT
                games.id, games.game_type_id, games.title, games.description, games.is_beta, g
ames.is_visible, games.cache_players_count, games.cache_ratings_json,
                games.cache_challenges_count_json, games.thumbnail_url, games.game_url, games.
introduction, games.language_ids_order, heroes.hero_name, companies.name as company_name,
                GROUP_CONCAT(levels.id) as level_ids, count(levels.id) as total_subtopics, gam
es.level_ids_order
            FROM games
            LEFT JOIN levels ON levels.game_id = games.id
            INNER JOIN heroes ON heroes.id = games.hero_id
            INNER JOIN companies ON companies.id = games.company_id
            WHERE games.id = ?
            GROUP BY games.id
            `, [game_id]
        );
    }
    try {
        let game_result = await databaseQueryModel.executeQuery(`Games Model | GetAdminGames (
${game_id})`, fetchAdminGamesquery);
        let fetchLanguageQuery = mysql.format(`SELECT id, name FROM languages`);
        let language_result = await databaseQueryModel.executeQuery("Games Model | GetAdminGam
es.fetchLanguage", fetchLanguageQuery);
        return {status: true, result: game_result, language_result: language_result};
    } catch(err) {
        return {status: false, message: "Failed to fetch admin games.", err: err};
    }
}
/* Function will fetch all games */
games.fetchAdminGames = async (req=undefined) => {
    return  await GetAdminGames(undefined, req);
}
```

Note how much better it is to read codes that are written using async/await?

Callback Exercises

As a way for you to get a bit more comfortable with callbacks, let's have you work on the following exercises.

1. Create a function that takes another function as its argument. Have the function execute the passed function.
2. Create a function that returns a function. Have the returned function be executed.
3. Create a function that takes two functions as its arguments. Randomly, either execute the first function or the second function.

Once you've completed the assignments above, then you're ready for the next step.

I have created additional challenges that require a knowledge of callback and put them in Hacker Hero's Advance Javascript Course: https://www.hackerhero.com/advanced-javascript-beta. Please start from the very first challenge and complete all challenges. The first five challenges will be a review of OOP concepts you've already learned. The next 11+ assignments will be focused on callbacks and other advanced features of Javascript.

Once you're done, take a screenshot of the map page (where it shows how many stars you've earned for each challenge). You will find at least 16 challenges in that Hacker Hero course.

Promise and Async/Await

As a way of practicing syntaxes for promise as well as async/await, let's have you build out a program that does the following:

Create a function called EmitRandomNumber(). In this function, after 2 full seconds (2000 ms), have it generate a random number between 0 to 100. If the random number generated is below 80, have it call that function again, up to 10 times, until the random number generated is greater than 80.

After the program is run, have it generate a log such as follows:

```
Attempt #1. EmitRandomNumber is called.
2 seconds have lapsed.
Random number generated is 35.
- - - - -
Attempt #2. EmitRandomNumber is called.
2 seconds have lapsed.
Random number generated is 76.
- - - - -
Attempt #3. EmitRandomNumber is called.
2 seconds have lapsed.
Random number generated is 53.
- - - - -
Attempt #4. EmitRandomNumber is called.
2 seconds have lapsed.
Random number generated is 85!!!
- - - - -
```

Note that the maximum number of attempts could be set as 10, so that the program doesn't take more than 20 seconds or so to complete, at maximum.

Note that doing above with traditional program such as C, PHP, Java, C#, or Python would be pretty straight forward. Doing this in Javascript will not be as easy (due to the asynchronous nature of Javascript). You'll soon learn more about this.

You'll do this in three different approaches:

1. Use traditional callbacks to accomplish the desired tasks.
2. Use promises this time.
3. Use async/await

Then, compare side by side all three approaches and see which one you feel most comfortable with. Unfortunately, for Javascript developers, as a lot of libraries use all three approaches (or combinations of the three), you will have to get familiar with all three approaches, although my recommendation is that by default, you should try to use async/await as much as possible.

OOP Design Patterns

There are lots of design patterns for Object Oriented Programming. To get some glance on what design patterns are, spend no more than 10 minutes browsing quickly here: https://en.wikipedia.org/wiki/Software_design_pattern

A lot of software design patterns are possible because of OOP (although not all design patterns require OOP).

Now, what we've taught you so far are the fundamentals. These fundamentals can be used to create some very complex design patterns to build things that are very complex.

It's important that you get yourself familiar with what design patterns are and how they relate to the industry today. I have taken time to write my perspectives on this at my medium channel: https://medium.com/@codesensei/solid-principles-are-they-as-important-as-they-used-to-be-20-years-ago-9a6a2c272fcd?sk=44a7a1d17cd663bcd89eda2677fc66a8

I do consider the article above to be one of the best articles I've written, so please read carefully and digest as much as you can. This will give you perspective on what's happening in the world today.

Some key take aways for you to remember are:

- there are lots of design patterns out there. You can spend your entire life time studying about these (and there are people who have devoted years of their life learning and exploring this area).
- using OOP and the fundamentals you've learned, you can create a lot of complex relations, including awesome data structures to efficiently sort, retrieve, update large amounts of information.
- Data structures such as Hash Table, Linked List, Binary Search Tree, Arrays, Dictionaries, Stacks, Queues, etc, are all ways of using Object Oriented Programming to store information efficiently, for optimizing specific activities. What I consider to be fundamental Data Structures are covered through the Hacker Hero's Advanced Algorithm and Data Structure Course, which I would highly recommend you do before you prepare yourself for a technical interview. But for now, do focus on the fundamentals as some of these concepts may be a bit too advanced for you, at this stage of your learning.
- these design patterns are especially important if you're building desktop applications, creating an operating system, creating a new programming language, or creating a new database engine from scratch.
- for web applications, there are only a few design patterns (such as MVC) that are used every day by web developers. Other complex design patterns are not as important for web developers although some basic principles such as SOLID is helpful especially as web applications become more complex.

Introduction to Data Structures

If you're ahead of schedule, please spend up to 5 hours going over Hacker Hero's Advanced Algorithm and Data

https://www.youtube.com/channel/UCFSr6LpI9k9kA6-wuVHfoMA/videos

Structure course. This will help you get more familiar with data structures and also better prepare you for technical interviews.